Fine-Grained Access Control Aware Multi-User Data Sharing with Secure Keyword Search

著者	ZHAO Fangming, NISHIDE Takashi, SAKURAI Kouichi
journal or publication title	IEICE transactions on information and systems
volume	E97.D
number	7
page range	1790-1803
year	2014-07
権利	(C) 2014 The Institute of Electronics, Information and Communication Engineers
URL	http://hdl.handle.net/2241/00122511

doi: 10.1587/transinf.E97.D.1790

IEICE TRANS. INF. & SYST., VOL.E97–D, NO.7 JULY 2014

PAPER

Fine-Grained Access Control Aware Multi-User Data Sharing with Secure Keyword Search*

Fangming ZHAO[†,††a)], **Takashi NISHIDE**[†††], ***and*** **Kouichi SAKURAI**[†], ***Members***

SUMMARY We consider the problems of access control and encrypted keyword search for cryptographic cloud storage in such a way that they can be implemented for a multiple users setting. Our fine-grained access control aware multi-user secure keyword search approach interdependently harmonizes these two security notions, access control and encrypted keyword search. Owing to the shrinkage of the cloud server's search space to the user's decryptable subset, the proposed scheme both decreases information leakage and is shown to be efficient by the results of our contrastive performance simulation.

key words: *cryptographic cloud storage, multiple users, access control, encrypted keyword search*

1. Introduction

To address users' concerns about confidentiality and privacy when using the cloud storage service, one common approach is adopting cryptographic techniques. Kamara *et al.* [11] proposed an architecture that combines several latest cryptographic primitives with the cloud storage, called *cryptographic cloud storage*. Even if this model will ease user's concerns about data leakage, it also introduces some new problems: because the encryption of data is not meaningful to the cloud servers, many useful data processing operations performed by cloud servers become infeasible.

In this paper, we consider a multi-user cryptographic cloud storage model to conveniently satisfy users' requirements of data confidentiality and privacy. A general use case of the electrical medical record (EMR) [1] based on this model is described as follows: a patient, Alice, wants to subscribe her EMR to the medical data center. The service allows Alice to share her health information and medical record with doctors from different hospitals and staffs from pharmacies or insurance companies. In order to protect her privacy, Alice wants to encrypt all her information, ensuring that even the employees of the data center or other unrelated doctors and staff members cannot know what is inside. Only authorized doctors are allowed to search and read her prior encrypted EMR and some of them are also authorized to update some items. These authorized doctors (perhaps belonging to different hospitals/departments) need to update related records in real time so that the latest EMR can be shared with all relevant persons. The read/update privilege management mechanism of the EMR must be independent of the data owner, Alice, once she creates the EMR access permission rule, because it is impossible to require a patient to be always available online to manage each data access of her EMR.

1.1 Challenging Issues

Even if many security protocols for cloud storage have been proposed, we find that some significant characteristics are still unsatisfied in the multi-user cloud storage environment. Below, we summarize several challenging issues.

- Several existing works adopt traditional or the latest cryptographic primitives for providing secure access control to the cloud storage. However, because it is difficult for a cloud server to differentiate writers and readers of each encrypted file, most schemes only consider a simple use case that the data owner creates the encrypted file for sharing with multi-users who are allowed to read but not to update the file, and only the owner is allowed to update the file. We call it *1-write-many-read*. Providing both write and read access permissions to multiple users can realize a more flexible access mechanism for cryptographic cloud storage. For example, after the data owner creates an encrypted file on the cloud, users who hold update access rights are allowed to update that file at a later time without help from the owner. We call such a mechanism the *owner-independent many-write-many-read* (*OI-MWMR*). To the best of our knowledge, no existing fine-grained access control protocols achieve the *OI-MWMR*.
- Most of the existing encrypted keyword search schemes ignore the access right of the user while performing the search algorithm because in general, it is difficult for the cloud server to distinguish the relationship between a user and numerous encrypted files only through an encrypted keyword (or called a trapdoor) he/she generated. Such an issue further brings two problems:

Manuscript received June 11, 2013.
Manuscript revised February 17, 2014.

[†]The authors are with Kyushu University, Fukuoka-shi, 819–0395 Japan.
[††]The author is with Toshiba Corporate Research & Development Center, Kawasaki-shi, 212–8582 Japan.
[†††]The author is with University of Tsukuba, Tsukuba-shi, 305–8577 Japan.

*A preliminary conference version of this paper appeared at [25]&[26]. Three main differences between the submitted manuscript and the preliminary works are listed as: definitions and proofs of security requirements, comparison with existing works, and performance analysis.

a) E-mail: zhao@itslab.csce.kyushu-u.ac.jp
DOI: 10.1587/transinf.E97.D.1790

  - Usually, not all users are allowed to read (decrypt) all the files on the cloud server. If each user is allowed to search keywords through all the files and obtains the result that includes those undecryptable ones, privacy information leakage may happen: the result tells whether a keyword is (or, not) related to any files, even if both the keyword and the file are encrypted.
  - Since a search algorithm is processed by the cloud server, the performance of cloud server is an essential issue. The performance can be improved much more if the cloud server only searches each keyword from a subset (e.g. decryptable files) but not from the whole storage.
- If we want to apply some traditional encrypted keyword search schemes (e.g. [7], [8]) to the multi-user cryptographic cloud storage setting, a naive approach is sharing the secret (search) key. However, sharing keys is generally not a good idea because it increases the risk of key exposure. Since a shared secret key must be changed if any user is no longer qualified to access the data, changing keys also results in re-generating all secure indexes. For the cloud storage with numerous users and files, this approach is not practical.

1.2 Our Contributions

In this paper, we study the access control and keyword search scheme of cryptographic cloud storage. We summarized main contributions of our work.

- We propose an access structure based reader/writer differentiation mechanism. Based on this idea, the *OI-MWMR* is successfully achieved and implemented to both of the file body and its secure indexes.
- We present an encrypted keyword search approach that we call fine-grained access control aware encrypted keyword search: the cloud server is only allowed to search an encrypted keyword over the user's decryptable data subset using the proposed *access structure computation*. As an additional contribution, a binary tree based file management approach is proposed to reduce the computation overhead and optimize the search efficiency. Two advantages of our approach over most existing works are shown by our security analysis and performance simulation:
  - Decreasing the information leakage from the keyword search process which is executed between users and the cloud server.
  - Being more efficient than existing works since the proposed method does not need to examine those unreadable files.
- Since each user uses a distinct secret key for his keyword search, the key update and the user revocation can be easily achieved without complicated processes of decryption and re-encryption of indexes.
- Several newly extended security requirements are defined for the multi-user model. The security of our scheme is successfully analyzed and proved based on them.
- *Attribute-based encryption (ABE)* is widely applied to the cloud for secure data sharing. However, few practical keyword search scheme is proposed for them because of the complex composition of its ciphertext. Our work gives a simple and practical solution for *ABE* based cloud storage.

2. System Models

We first introduce the entities involved in our scheme, then we identify two important functional concepts.

2.1 Entities

- **Cloud Server**: The main responsibility of the cloud server is to store and to process encrypted data according to authorized users' requests. The cloud server is modeled as *honest-but-curious* in our scheme. Cloud servers are assumed to be semi-trusted, which mean servers are honest to save user's file and to perform data operations requested from authorized parties.
- **Trusted Authority (TA)**: TA is a fully trusted third party. Firstly, it is responsible for managing all attributes and their related cryptographic keys. Secondly, it manages user's enrollment and revocation for the proposed scheme. The setup process of keys for users and the cloud storage server is operated by TA.
- **Data Owner**: A data owner first creates (encrypts) data for sharing and secure indexes for keyword search, defines the access privileges policies, and then set them up to the cloud server.
- **Users**: There are multiple users with different access privileges which map to numerous files on the cloud server. Except for the data owner, we define two kinds of users for each file, readers and writers. Readers who have the decryption right and can read. Writers who have both decryption and encryption rights, can read/update files (including update secure indexes). Both readers and writers are able to require the cloud to perform the keyword search to retrieve encrypted files.

2.2 Functionality Goals

- **Fine-grained access control with OI-MWMR**:
  It facilitates specifying both read and write access rights to each file for a set of users in terms of their attribute set. The *OI-MWMR* (*owner-independent many-write-many-read*) means that for each file on the cloud, there may exist multiple writers and readers respectively. Writers are allowed to update a file in a secure manner without any help (e.g. a real-time authorization) from the data owner.

- **Fine-grained access control aware keyword search**: Let n be the number of encrypted files on the cloud, and a user, u, wants the server to search the files that contain a keyword w. m ($m \leq n$) is the file number on the cloud that can be decrypted by u. This functionality requires that the cloud executes the keyword search after narrowing the search scope from n to m by being aware of access privileges of u.

3. Security Requirements and Assumption

3.1 Security Requirements and Definitions

We integrate and exploit several security properties from [2], [6], [23], [25], [26] to summarize security requirements for the proposed scheme under the multi-user cloud model.

- **Impersonation Resistance**. Traditionally, impersonation resistance requires that an adversary cannot authenticate itself as a legitimate user to any honest entity. In our model, both readers and writers are legitimate users. If a reader successfully impersonates a writer, then he can modify the corresponding data on the cloud server which includes: encrypted data, secure indexes, and access policies. Here, this property is extended defined as no readers can impersonate writers to illegally update any data on the cloud server.
- **Inaccessible Information Invisibility**. Our work first defines this security property for the encrypted keyword search scheme. It requires no user is allowed to access the data which is not decryptable for him according to his access privilege. In our scheme, it requires a user cannot get information (such as *Search(w)* $\stackrel{?}{=} \phi$: whether the search result for keyword w is null) from search results over his undecryptable files.
- **Query Privacy**. Query privacy is a common security requirement for all encrypted keyword search schemes. This security notion mainly considers the amount of information leakage (i.e. information that directly relates to plaintext of data and keywords, corresponding secret keys) to the cloud server regarding user queries. In other words, apart from the information that can be acquired via observation and the information derived from it, this notion requires no other information should be exposed. Let $\{Q_1, Q_2, \ldots, Q_t\}$ be a sequence of t queries, and $W_t = \{w_1, w_2, \ldots, w_t\}$ be the corresponding queried keywords. Let $A_t = \{a_1, a_2, \ldots, a_t\}$ be the corresponding replies, where $t \in \mathbb{N}$ is polynomially bounded. We define the view V_t of an adversary over the t queries as the transcript of the interactions between the server and the involved query issuers. V_t contains the ciphertext data CT and the secure indexes I, queries Q_t and the replies A_t. Let T_t be the information that we allow the adversary to obtain, which includes the results A_t of t queries, the identifying information (such as its hard disk position or its memory location) of each data referred in A_t and the issuer of Q_t. Finally, a simulation based definition of query privacy is formally presented as follows:

  Definition 1: *Query Privacy*. An encrypted keyword search protocol achieves query privacy if for all data D, $t \in \mathbb{N}$, and all *PPT* algorithms $\mathcal{A}$, there exists a *PPT* algorithm (simulator) $\mathcal{A}^*$, such that for all V_t and T_t, for any function f:
  $|\Pr[\mathcal{A}(V_t) = f(D, W_t)] - \Pr[\mathcal{A}^*(T_t) = f(D, W_t)]| < \nu(k)$

  NOTE: k is the security parameter. A real-valued function $\nu(k)$ is negligible if for any polynomial $p > 0$ there exists a $k_p > 0$ such that $\nu(k) < 1/p(k)$ for all $k > k_p$.
- **Query Unforgeability**. This property is only applicable to the multiple users setting: it requires a dishonest user cannot generate a legitimate query on behalf of another (valid) user. For a user u from the valid user set U ($u \in U$) and a keyword w, we define u's legitimate query set as $\hat{Q}_u = \{Q_u(w), Sig(Q_u)\}$. $Sig(Q_u)$ is the signature on $Q_u(w)$ if $Q_u(w)$ is indeed generated by u's secret key K_u, and $Sig(Q_u)$ is generated by u's signing key SK_S. Query unforgeability is defined based on a game between an adversary and a challenger. Let $\hat{u}$ be the target user of the adversary $\mathcal{A}$. In $\mathcal{A}$'s game, the challenger simulates the protocol which allows $\mathcal{A}$ to obtain queries on keywords of her choices with respect to user $\hat{u}$. Specifically, $\mathcal{A}$ first picks her target user $\hat{u}$ and is given keys of the remaining users, say, $u \in U \backslash \hat{u}$. Then $\mathcal{A}$ queries the oracle Φ which returns a set $\{Q, Sig(Q)\}$ at her will with the restriction that the number of queries is polynomial-bounded. Let $Q'_{\hat{u}} = \{Q'_{\hat{u}}, Sig(Q'_{\hat{u}})\}$ denote the set of $\hat{u}$'s queries and signatures obtained by $\mathcal{A}$. $\mathcal{A}$ wins the game if and only if the generated $(Q, Sig(Q)) \in Q_{\hat{u}} \backslash Q'_{\hat{u}}$. The advantage of $\mathcal{A}$ against query unforgeability is defined as the probability of her winning the game.

  Definition 2: *Query Unforgeability*. An encrypted keyword search protocol achieves query unforgeability if for any $\hat{u} \in U$, and for all *PPT* algorithms $\mathcal{A}$:
  $\Pr[(Q, Sig(Q)) \in Q_{\hat{u}} \backslash Q'_{\hat{u}}: Q \leftarrow \mathcal{A}^{\Phi}(\{K_u \mid u \in U \backslash \hat{u}\}) \wedge Sig(Q) \leftarrow \mathcal{A}^{\Phi}(\{SK_u \mid u \in U \backslash \hat{u}\})] < \nu(\kappa)$
- **Revocability**. Revocation is an indispensable property for all multi-user schemes. TA is responsible for managing users' identities in our scheme. If a user is no longer allowed to access files on the cloud, one of the most important tasks of TA is revoking his search capability. Since the incapability of searching the indexes is implied by the incapability of distinguishing them, we define revocability based on the index indistinguishability.
  An adversary's advantage in attacking revocability is defined as her winning probability in the following game. The adversary $\mathcal{A}$ runs in following two stages:
  - $\mathcal{A}_1$. In the first stage, $\mathcal{A}_1$ acts as an authorized user and is allowed to access the oracle Φ as described in Definition 2. At the end of this stage,

$\mathcal{A}_1$ chooses two new keywords w_0 and w_1 which have never been queried thus far. Let *state* represent the knowledge $\mathcal{A}_1$ gains during the first stage.
– $\mathcal{A}_2$. In the second stage, $\mathcal{A}_2$ is revoked and given the index of w_b where a coin $b \in \{0, 1\}$ is tossed. Finally, $\mathcal{A}_2$ outputs a bit b' (as its guess for b).

$\mathcal{A}$ wins the game if and only if $b' = b$.

Definition 3: *Revocability*. An encrypted keyword search protocol achieves revocability if for all *PPT* algorithms $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$:

$|\Pr[b' = b$: $(state, w_0, w_1) \leftarrow \mathcal{A}_1^{\Phi}$;
Revoke$(\mathcal{A})$;
$b \in \{0, 1\}$, $I(w_b) \leftarrow$ **BuildIndex**$(w_b, CK_{U_{id}})$;
$b' \leftarrow \mathcal{A}_2(state, I(w_b), w_0, w_1)$;
$] - 1/2 | < \nu(\kappa)$

Remark. This definition of revocability based on the index indistinguishability addresses the revocation of the keyword search capability. The purpose of our definition is different from the attribute revocation (e.g. [12]) which aims to deprive users' read access privilege corresponding to those revoked attributes. If attribute revocation happens in our system, it requires the update of ciphertext and its signature, and (possibly) users' secret keys. All these should be executed together with TA, the cloud server, and data owners, etc. The discussion of attribute revocation is out of the scope of this paper. However, we argue that both (*i*) keyword search capability and (*ii*) data access privilege will be heavily influenced by the attribute revocation in our system, and this problem is considered as one of our future research.

3.2 Assumption

We assume that the *user-server* collusion is not included in our adversarial model. Although this assumption is quite strong, it is a practically reasonable assumption which is also utilized in [2], [23]. In our scheme, secret keys and the plaintext of a trapdoor are still kept secure even if such an active attack (*user-server collusion*) is launched. However, from a technical perspective, the attack is able to comprise most search schemes in another form: the server can always compare the access patterns between a target user and the colluding user. Furthermore, illegal file-updates (coming from the malicious users) will be permitted. All communication between any two parties is also assumed secure under TLS/SSL in the network/transport layer.

4. Technical Preliminaries

Our scheme builds on the work by Bethencourt *et al.*, Ciphertext-policy attribute-based encryption (CP-ABE) [3], and Maji *et al.*, Attribute-based signature (ABS) [14]. Only a conceptual introduction is given here, please refer to Appendix A for a more detailed algorithm description.

CP-ABE is one of the latest public key cryptography primitives for secure data sharing. A user's private key will be associated with an arbitrary number of attributes expressed as strings. When a party encrypts a message, they first specify an associated access structure over attributes. Users will be able to decrypt a ciphertext if their attributes satisfy the ciphertext's access structure.

ABS is a versatile primitive that allows a party to sign a message with fine-grained control based on its attributes. More specifically, the signer, who possesses a set of attributes, can sign a message with a predicate that is satisfied by his attributes. It ensures that only a signer can generate a signature if her attributes satisfies the predefined predicate. E.g. in a hospital, only a doctor who satisfies the $T_{self-sign}$ (Fig. 2) can issue an official certificate of heart diseases.

5. Concrete Construction

We introduce concrete construction of the proposed scheme in this section. We consider such a scene: after the data owner creates encrypted files on the cloud, other users (readers/writers) can securely read/write, and retrieve interested data using the keyword search scheme which is executed by the cloud server while considering their access right. Note that a novel and important characteristic of our scheme is that both encrypted file body and their encrypted keyword indexes support the *many-write-many-read*, which is a key contribution compared to other existing schemes.

5.1 Design Concept

5.1.1 Fine-Grained Access Control with OI-MWMR

One of the most important access control characteristics in the multi-user cryptographic cloud storage model is the ability of separating readers and writers of a file. We first propose an access structure based reader/writer differentiation mechanism which achieves the *OI-MWMR*. The file owner first decides two access structures $T_{decrypt}$ (used in *CP-ABE*) and $T_{update-sign}$ (used in *ABS*), see Fig. 1. Then he sets (uploads) the $T_{update-sign}$ with its encrypted file to the cloud server. For other users (writers) who want to update the file on the cloud server at a later time, they must possess attribute sets described in $T_{update-sign}$. Note that we do not need to differentiate writers and readers at the individual-user level but at an attribute level. The latter is much better optimized than the former whose management complexities may increase linearly upon the number of users, but the latter will not. As an example in Fig. 1, a patient with heart disease creates his EMR access policy for sharing with others: users who satisfy attributes tree "cardiologist"$\vee$"nurse"$\vee$"insurance staff" are allowed to read (decrypt) his EMR; users who satisfy attributes structure "nurse"$\wedge$"insurance staff"$\vee$"cardiologist" are not only allowed to read but are also allowed to update.

Note that our scheme is different from another ABE based solution, signcryption [19], [24], which integrates ABE and ABS together (it only requires one single access structure). Since the decryption and verification must be ex-

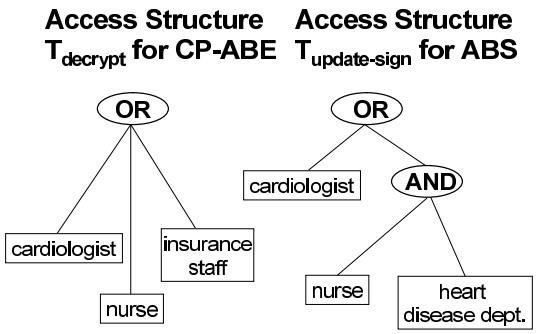

Fig. 1 Access structure I (access tree).

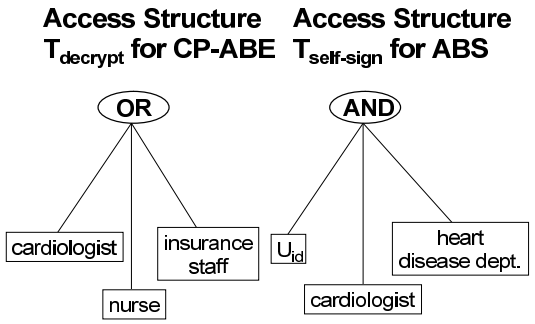

Fig. 2 Access structure II (access tree).

ecuted simultaneously, it thwarts their scheme from differentiating readers and writers.

5.1.2 Fine-Grained Data Access Control Aware Keyword Search

Access control needs to be enforced before the cloud server searches a keyword, and, a user is not allowed to search through data which is not decryptable for him. We construct our proposal based on the cloud infrastructure as described in Sect. 5.1.1 and a query protocol of Bao *et al.* [2], [23]. We stress that it is not sufficient to use these schemes *as-is* to achieve functionality goals defined in Sect. 2 because it is difficult for the cloud server to distinguish the relationship between a user and numerous encrypted files only through an encrypted keyword (or called a trapdoor) he/she generated. Our approach, *access structure computation* (Definition 4), successfully solved this challenge issue and achieved the functionality goal.

To realize our idea, we take advantage of the access structures of *CP-ABE* and *ABS* to (*i*) allow the cloud server to focus on the user's decryptable file group; (*ii*) make a user prove to the cloud server that he really holds those attributes for decryption before his query is executed. More specifically, (*i*) is achieved by comparing the $T_{decrypt}$ of *CP-ABE* with the $T_{self-sign}$ of *ABS* which is generated by the user. (*ii*) is achieved by generating an *ABS* using the *AND* of all his attributes: $T_{self-sign} = \{U_{id} \wedge Att_1 \wedge Att_2 \wedge Att_3 \ldots\}$ (Note, U_{id} is the user's ID and it is considered as a special attribute). The result of signature verification shows whether the user holds those attributes as he claims. An example is given in Fig. 2: a doctor can prove the possession of his attributes "*cardiologist*" $\wedge$ "*heart disease dept.*" $\wedge$ "U_{id}" by generating an *ABS* with the $T_{self-sign}$. If the verification succeeds, the cloud storage server clarifies whether the user can decrypt a file by checking: $T_{self-sign} \models T_{decrypt} = 1$ or 0, "$\models$" is formally defined:

Definition 4: Let T_1 (e.g. $T_{self-sign}$) and T_2 (e.g. $T_{decrypt}$) be two access trees (also called access structure) in attribute-based cryptosystems. $T_1 \models T_2$ is an access structure computation that outputs 1 or 0, where 1 means that at least one attribute (e.g. "*cardiologist*") described in T_1 meets the requirement of T_2. 0 means no such attributes exist in T_1 which can satisfy T_2[†].

The proposed revocation mechanism enables the system administrator to dynamically and efficiently revoke the future's search capability of a malicious user. We consider that such a countermeasure (revoking the search capability) as a reasonable and cost-effective method against malicious users in our system. First, a revoked user, who holds decryption access privilege of some files, cannot get useful data through a wiretapping because all the communication is protected under TLS/SSL in the network/transport layer. Second, a revoked user may indeed decrypt some files with his old keys if he successfully intrudes into the cloud server. Sahai *et al.* [18] proposed a solution of attribute revocation which enables the "*ciphertext delegation*" instead of a simple "decrypt then re-encrypt" to prevent such a threat. Such protocols would solve this threat where revoked users still can access previously decryptable ciphertexts with their old keys. This issue is out of the scope of this paper and could be considered in our future work.

5.2 Proposed Scheme

The proposed scheme consists of a tuple of algorithms {*Setup()*, *Create()*, *Write()*, *Query()*, *Search()*, *Read()*, *Update()*, *Revocation()*}. Next, we give a detailed description.

1. **Setup()**: The initialization algorithm *Setup()* is run by TA to set the key materials for processing both the file and its query related data.
   a. TA outputs cryptographic keys, (PK_E, SK_E, PK_S, SK_S), for appropriate users according to their attributes. (PK_E, SK_E) are public/private keys for *CP-ABE* based file encryption. (PK_S, SK_S) are public/private keys for *ABS* based file signature generation. Note: users who possess different attributes set will hold different keys, please refer Appendix A for the detail of key generation.
   b. TA outputs keys to users for their encrypted keyword search. TA takes as input the security parameter 1^k and outputs its unique master secret key $K_{msk} \in \mathbb{Z}_p$ and the key pair $\{K_{U_{id}} \in \mathbb{Z}_p, CK_{U_{id}}\}$ for each user whose user ID is U_{id}, where $CK_{U_{id}} =$

[†]In Sect. 7, we also propose a binary tree based file management approach which can greatly reduce the server's computation overhead in the access structure computation.

$g^{K_{msk}/K_{U_{id}}}$ is a complementary key for a user. $K_{U_{id}}$ is only distributed to the user as his secret key. $\{U_{id}, CK_{U_{id}}\}$ are only sent to the cloud server.

2. **Create()**: This algorithm includes two steps: *CreateFile()* creates an encrypted file with signature; and *BuildIndex()* builds its indexes of selected keywords.

   a. **CreateFile()**: The data owner first encrypts a file for sharing with other users. The encryption is based on *CP-ABE*. The decryption policy in the CT is described by the access structure $T_{decrypt}$. The ciphertext CT of a file M is generated as:

   $$CT = Enc(PK_E, M, T_{decrypt})$$

   The owner then generates the signature of CT. He hashes the CT and then signs it by the *ABS*. To prevent the replay attack, we insert a version number tag n to the *ABS*. The server later only accepts valid updates if the new version number tag n' satisfies $n' = n + 1$. Later, the user will confirm the latest n of the CT from the cloud server before she generates the signature SG.

   $$SG = Sign(PK_S, SK_S, h(CT)||n, T_{update-sign})$$

   b. **BuildIndex()**: The algorithm is run by the owner and the cloud server interactively. This algorithm outputs a secure index $\mathrm{I}(w_i)$ for a keyword w_i from $\{w_1, w_2, \ldots\}$. The data owner first uploads the $\{U_{id}, h(w_i)^r\}$ to the cloud server. $h()$: $\{0,1\}^* \rightarrow \mathbb{G}_0$ is a collision resistant hash function and $r \in \mathbb{Z}_p$ is a random number. After receiving the request, the cloud server calculates the Cap_w for each w_i and then sends it back to the data owner.

   $$Cap_w = e(h(w_i)^r, CK_{U_{id}})$$

   The data owner can build the index for each w_i as $\mathrm{I}(w_i)$. k is the key for *HMAC* and $R \in \mathbb{Z}_p$ is a random number.

   $$I(w_i) = [R, HMAC_k(R)],\ k = h(Cap_w^{K_{U_{id}}/r})$$

3. **Write()**: The owner writes (or uploads) both the encrypted file (with signature) and its secure indexes to the cloud server. Note the access structure of ABS, $T_{update-sign}$ (which is transmitted separately with the SG), allows the cloud server to differentiate readers and writers at a later time. Finally, $\{CT, SG, n, I(w_i), T_{update-sign}\}$ are written to the cloud server.
4. **Query()**: For a specific keyword w_i, the user first generates a trapdoor $Q(w_i)$, then he generates an ABS, $Sig(Q(w_i))$.

   $$Q(w_i) = h(w_i)^{K_{U_{id}}},$$
   $$Sig(Q(w_i)) = Sign(PK_S, SK_S, h(Q(w_i)), T_{self-sign})$$

   Note that the $T_{self-sign}$ is made by all of the user's attributes including the user's ID: $T_{self-sign} = \{U_{id} \wedge Att_1 \wedge Att_2 \wedge Att_3 \ldots\}$. The signature *ABS* shows that the user certainly possesses a set of attributes from the authority as he/she declared in the access tree $T_{self-sign}$. The cloud server verifies the user's attributes by public keys from TA. In this step, the user sends $\{U_{id}, Q(w_i), Sig(Q(w_i))\}$ to the cloud server.
5. **Search()**: After receiving the query, the server first checks the complementary key $CK_{U_{id}}$ by the user ID, U_{id}. If the U_{id} is valid, the server confirms the user's decryptable file group by: (*i*) Verify attribute set of the user as described in $T_{self-sign}$ by the *ABS*-verification,

   $$Verify(PK_S,\ h(Q(w_i)),\ T_{self-sign},\ Sig(Q(w_i))) \stackrel{?}{=} true$$

   The verification key PK_S is published by TA. If the *ABS* verification result is true, the user's attributes as he/she declared in the $T_{self-sign}$ are confirmed. (*ii*) Using $T_{decrypt}$ from CT, the cloud server can confirm the search scope S as the following procedure:

   $S = Null;$
   $for(i = 0; i < n; i++)\{$
   //i: index number; n: total number of files.
   $\quad if((T_{self-sign} \models T_{decrypt}[i])! = 0)$
   $\quad S = S \cup i;\}$
   $return\ S;$

   Then the cloud server performs the the keyword search only over the scope S. It first computes $k' = e(Q(w_i), CK_{U_{id}})$, and then checks each index of the data CT in the scope S as: $HMAC_k(R) \stackrel{?}{=} HMAC_{k'}(R)$. Finally, the server sends the search result to the user.
6. **Read()**: Using the result of the *Search()* step, a valid user can get the target files and read. The *Read()* algorithm first verifies the SG with $T_{update-sign}$ and corresponding public keys PK_S from TA.

   $$Verify(PK_S, h(CT)||n, T_{update-sign}, SG) \stackrel{?}{=} true$$

   If the verification is successful and the user's attributes U satisfies $T_{decrypt}(U) = 1$, then he can decrypt CT and gets the plaintext of M.

   $$M = Decrypt(CT, SK_E)$$

7. **Update()**: If a user holds writer's access right (attributes), then he can update a file.

   a. Encrypt M_1 to CT_1.

   $$CT_1 = Enc(PK_E, M_1, T_{decrypt_1})$$

   b. Make a new SG_1 with a new version number $n' = n + 1$.

   $$SG_1 = Sign(PK_S, SK_S, h(CT_1)||n', T_{update-sign})$$

   c. Upload $\{CT_1,\ SG_1,\ n',\ T_{update-sign}\}$ to the cloud server as the *Write()* phase. Cloud storage server will first check the version number tag n', then verify the SG_1 as depicted in the *Write()*. Finally, the cloud server accepts or rejects the update request according to the *ABS* verification result.

Remark. The differences between *Update()* and *Create()* can be clarified as: (*i*) In each *Update()*, the cloud server needs to check the ABS (e.g. SG_1) using the $T_{update-sign}$, where such a process is not required in *Create()*. (*ii*) In *Update()*, the writer can update the $T_{decrypt}$ (e.g. change the attribute set which is initialized in *Create()*) to revoke/grant any attribute(s) at the specific file-level. (*iii*) In *Update()*, the writer is able to update the secure indexes which are initialized in *Create()*.

8. **Revocation**(): This algorithm remove a user's search ability. TA and the cloud server manage all users' pair $\{U_{id}, CK_{U_{id}}\}$. To revoke someone, TA just instructs the cloud server to delete the entry from the user list L: $L = L \setminus \{U_{id}, CK_{U_{id}}\}$, then that user is no longer able to search the cloud storage.

6. Security Analysis

We analyze the security of our proposed scheme, and in particular we show that the proposed scheme satisfies general security requirements described in Sect. 3.

Impersonation Resistance. Readers and writers have different privileges. If a reader successfully impersonates a writer, then he can illegally modify the corresponding data on the cloud server which includes: encrypted data, secure indexes, and access structures. The policy to differentiate writer with readers is defined as $T_{update-sign}$. Cloud server clarifies writers and readers based on the result of *ABS* verification. Both readers and other unauthorized users cannot impersonate writers' privileges because they cannot forge the *ABS* of writers. Consequently, the unforgeability of *ABS* ensures the impersonation resistance of our scheme.

Inaccessible Information Invisibility. Information leakage from search results needs to be considered when designing protocols for multi-user cryptographic cloud storage. In our scheme, by implementing and exploiting access structure from attribute-based cryptosystems, the cloud server only performs the keyword search on the user's accessible file subset. The result of $T_{self-sign} \models T_{decrypt}$ shows the cloud server whether a file is decryptable to the user without exchanging any secret key beforehand. As a result, the output of *Search()* will not involve redundant information (e.g. Whether the cloud holds any files that contains the same keyword). Our scheme achieves the property of inaccessible information invisibility.

Query Privacy. Our protocol achieves Definition 1 in the following theorem, the proof is given in Appendix B.

Theorem 1: The proposed encrypted keyword search scheme achieves query privacy in Definition 1 if *HMAC* is an unforgeable MAC, $h()$ is a pseudorandom function, and *CP-ABE* is secure.

Query Unforgeability. Our protocol achieves Definition 2 in the following theorem, the proof is given in Appendix B.

Theorem 2: The proposed encrypted keyword search scheme achieves query unforgeability in Definition 2 if ABS is an unforgeable signature scheme.

Revocability. Our protocol achieves Definition 3 in the following theorem, the proof is given in Appendix B.

Theorem 3: Our protocol achieves revocability in Definition 3 if *HMAC* is a preimage resistant *MAC* scheme.

7. Discussion and Performance Analysis

We first give a discussion of our schemes by comparing with several latest existing works. Then, we analyze the performance of our scheme in terms of the computation overhead and search efficiency.

7.1 Discussion

Several existing works close to ours have been proposed recently. Works of [9], [12], [22] are related to the access control of cryptographic cloud storage, and [2], [4], [6], [13], [20], [21], [23] are related to the multi-user encrypted keyword search schemes.

In Wang *et al.* [22], an owner's data is encrypted block-by-block using the symmetric key cryptography. A binary-key tree is constructed over the block keys to reduce the number of keys given to each user, and all binary-key trees for all files must be managed by files' owners. If owners want to share their file with other users, they must distribute the binary-key tree to all users individually. Users' read and write rights are not separable: valid users can only read (or called decrypt) files but cannot update the original files. Ion *et al.* [9] and Li *et al.* [12] adopted the *ABE* (*CP-ABE* or *KP-ABE*) for data encryption to achieve the fine-grained access control. Since their works and our work all try to realize fine-grained cryptographic cloud storage by the help of attribute based cryptosystems, we compare the computation complexity of these three works, and the results are summarized in Table 3. Ion *et al.* [9] describes a secure publish/subscribe framework in which publishers (owners) can share encrypted information with subscribers (readers) with the help of untrusted brokers (storage servers), a *1-write-many-read* scheme. The work of Li *et al.* [12] also describes a fine-grained data access control protocol for sharing personal health records in the cloud storage. Multi-authority *KP-ABE* of Chase *et al.* [5] is used for providing data confidentiality and fine-grained data access control. Compared with our scheme, Li *et al.* [12] considered a different set of requirements of the write accesses to the cloud: a time-limited write permission. To update a file, the user must first contact the owner (who must be online to reply) for a one-time individual authorization. The owner generates a signature with a specific valid period, and then encrypts the time-limited signature and the time information by a public-key encryption algorithm (Details of signature and public-key encryption algorithms are not specified, and we use C_{sign} and C_{enc} to denote their computation cost for a comparison in Table 3). Their update access control frequently requires

Table 1 A security functional comparison with existing schemes ("-" means out of the protocol design scope).

Security Mechanisms	Access Control			Multi-User Encrypted Keyword Search			
Security Characteristics	Fine-grained	Many-Write -Many-Read	Owner-indep. file update	TA/Owner indep. trapdoor gen.	Access right aware search	Revoc. without re-enc, re-index	Multi-U. updatable secure index
Ion *et al.* [9]	*yes*	*no*	*no*	-	-	-	-
Li *et al.* [12]	*yes*	*yes*	*no*	-	-	-	-
Wang *et al.* [22]	*no*	*no*	*no*	-	-	-	-
Bao *et al.* [2], [23]	-	-	-	*yes*	*no*	*yes*	*no*
Boneh *et al.* [4]	-	-	-	*no*	*no*	*no*	*no*
Curtmola *et al.* [6]	-	-	-	*yes*	*no*	*no*	*no*
Li *et al.* [13]	-	-	-	*no*	*no*	*no*	*no*
Tomida *et al.* [20]	*no*	*no*	*no*	*no*	*no*	-	*no*
C. Wang *et al.* [21]	*yes*	*no*	*no*	*yes*	*yes*	-	*no*
Our Scheme	*yes*	*yes*	*yes*	*yes*	*yes*	*yes*	*yes*

the owner's help. Moreover, different from our scheme, [12] discussed a different revocation, attribute revocation, which revokes the read access privilege. Consequently, neither of these schemes achieves the *OI-MWMR* for multi-user cryptographic cloud storage.

Works of [2], [4], [6], [13], [20], [21], [23] considered the multi-user encrypted keyword search scenario. Bao *et al.* [2], [23] is one of the design bases of our scheme. Their scheme allows each user to possess a distinct secret key for generating the trapdoor respectively. The key advancement of our scheme over theirs is that our scheme realize the fine-grained access control aware search approach and the multi-user updatable secure index. Boneh *et al.* [4] presented a scheme for searching on encrypted data using a public key system that allows mail gateways to handle email based on whether certain keywords exist in the encrypted message. In their work, asymmetric keys allow multiple users to encrypt data using the public key, but only the user who has the private key can search and decrypt the data. Sharing the unique private key with multi-users is one solution, however, in this case, the user revocation becomes prohibitively expensive because all queries are generated from the same key, and to revoke the key means not only to re-generate all the indexes but also to re-distribute a new key to all non-revoked users. Tomida *et al.* [20] proposed a searchable encryption scheme based on the identity based encryption. This scheme requires the owner generates a number of indexes for each search respectively, and each index is only restricted to a specific user's ID. In other words, before searching a keyword, a searcher needs to request the data owner to generate an individual index set on the cloud for his personal use only. Obviously, the management cost of personal indexes increase linearly with the number of searchers. Because their scheme searches a trapdoor through a predetermined index set which is manually assigned by the owner but not automatically determined by the searcher's read access right, it does not satisfy the requirement of access right aware search. Also, revocation is not considered in their scheme. Finally, [4], [20] are highly dependent on the existence of data owner, and users cannot execute the keyword search without the help of the owner (e.g. when he is offline).

Curtmola *et al.* [6] partly solved the multi-user problem using broadcast encryption. The set of authorized users share a secret key r (which is used in conjunction with a trapdoor function). Only people who know r will be able to access/query the data. A user can be revoked by changing r, and using broadcast encryption [15], [16] to send the new key r' to the set of authorized users. The revoked users do not know r', and hence cannot search. Li *et al.* [13] proposed authorized private keyword search (APKS) over encrypted data for multi-user cloud storage using the Hierarchical Predicate Encryption (HPE). In their construction of privacy aware search, capabilities (trapdoors) were distributed by a Trusted Authority (TA) or a Local TA (LTA). So, the trapdoor distribution is obviously a cumbersome task. C. Wang *et al.* [21] gave a keyword search encryption and some properties of their scheme appear similar to ours. They integrate the symmetric key predicate encryption into KP-ABE, which provides the keyword search with the property of fine-grained access control. Their scheme allows an owner to share his data with authorized readers, and these readers are also allowed to search the cloud according to their access right. The shared data (including its indexes) is not allowed to be updated by other users, and revocation is not supported. Since indexes are integrated into the ciphertext data, it is difficult to update the data or indexes separately. If the owner wants to add/remove an index, the only way is to reconstruct the whole ciphertext data. Consequently, works of keyword search schemes [4], [6], [13], [20], [21] do not support the (multi-user) update of indexes.

Table 1 gives a comparison between these existing schemes and our scheme. The comparison is functionality-classified in two folds: access control and encrypted keyword search. Access control includes three security characteristics, *Fine-grained*, *Many-Write-Many-Read*, *Owner-independent file update*. Encrypted keyword search includes *TA/Owner independent trapdoor generation*, *Access right aware search*, *Revocability without re-encryption and re-index*, *Multi-user updatable secure index*.

7.2 Performance Analysis

We first analyze the computational cost of the file body processing on the client side in Sect. 7.2.1. Then, we give a contrastive performance simulation to show the effective-

Table 2 Notations.

Notation	Description
E_0	Cost of exponentiation operations in $\mathbb{G}_0$
E_1	Cost of exponentiation operations in $\mathbb{G}_1$
L	Cost of bilinear pairing
p	Prime order of $\mathbb{G}_0$ and $\mathbb{G}_1$
U	The attribute set in the access structure (tree)
l, t	The matrix $\{l \times t\}$ of the monotone span program which is converted from its corres -ponding access structure

ness and efficiency of our fine-grained access control aware approach in Sect. 7.

7.2.1 Performance Analysis of Access Control Mechanism

As described in our proposal, the following steps, which process the body of a file for access control, are fully processed on the user's client side: (*i*) *CreateFile()*, (*ii*) *Read()*, (*iii*) *Update()*. For analyzing the computation complexities of each process which includes several cryptographic operations such as CP-APE and *ABS*, we use the following notations in Table 2.

The computation overhead generated from processes (*i*) and (*iii*), *CreateFile()/Update()*, are actually the same, which include two operations, one operation of *CP-ABE Encryption* and one operation of *ABS-Sign*. In terms of the computation details of *CP-ABE* and *ABS* that are described in Sect. 4, the user's computation costs of the *CP-ABE Encryption* and the *ABS-Sign* utilized in our proposed scheme both grow linearly with the size of access structure's matrix $\{l \times t\}$. These costs are mainly generated from the exponentiation operations in $\mathbb{G}_0$ and $\mathbb{G}_1$.

The computation overhead generated from the process (*ii*), *Read()*, also includes two operations, *CP-ABE Decryption* and *ABS-Verification*. In terms of the computation details in Sect. 4, the user's computation cost of the *CP-ABE Decryption* grows linearly with the number of his attributes which satisfy the access structure. More precisely, the cost is mainly generated from the exponentiation operations in $\mathbb{G}_1$ and paring computations. The user's computation cost of the *ABS-Verification* is also generated from the paring computations and exponentiation operations in $\mathbb{G}_0$. This cost also grows linearly with the size of access structure's matrix $\{l \times t\}$. The computation complexities of cryptographic operations which are included in the three main steps executed on the user's side are summarized in Table 3.

We make performance estimations for each process step: *CreateFile(), Read(), Update()* to show the feasibility of the proposed access control scheme. Basically, processing time of *CP-ABE* and *ABS* is dependent on the computations of paring and exponentiation, so we make the estimations based on the processing time of paring and exponentiation. The criterion of our estimation is based on the result of Guillevic [10], which implements the pairing over a prime-order elliptic curves on a computer with 2.6 GHz Celeron 64 bits CPU, 1 GB RAM and Ubuntu 10.04.4 LTS OS. On their implementation, one pairing takes *5.05ms*, and one exponentiation takes *5.16ms*. Assume N is the number of attributes in the access structure. From details of cryptographic operations described in Sect. 4, the *CP-ABE Encryption* is mainly composed of *(2N+2)* exponentiations; the *CP-ABE Decryption* is composed of *(2N+1)* parings and N exponentiations. In the other side, the processing of *ABS* is a little complicated than the *CP-ABE*. The *ABS-Sign* is composed of *(5N+3)* exponentiations; the *ABS-Verification* process, in the maximum case, is composed of *(N+2)* parings and *(2N+1)* exponentiations. As an example of $N = 10$, let t_{create}, t_{read} and t_{update} denote the processing time of *CreateFile(), Read(), Update()*. According to the criterion from [10], the processing time will approximately be: $t_{create} = 387ms$, $t_{read} = 327ms$, $t_{update} = 387ms$. Compared with files download and upload time, we think the performance of the proposed scheme is reasonable for cryptographic cloud storage.

7.2.2 Performance Analysis of Access Control Aware Keyword Search Scheme

We first give a binary tree based file management approach for the proposed access structure computation which greatly reduces the cloud server's computation overhead. Then, we give a theoretical simulation to show the effectiveness and efficiency of our scheme.

The result of the access structure computation $T_{self-sign} \models T_{decrypt}$ shows whether the encrypted file with an access structure $T_{decrypt}$ can be decrypted by a user with attributes described in her $T_{self-sign}$. Assume the number of encrypted files on the cloud server is n, then in a naive implementation, the cloud server must execute $T_{self-sign} \models T_{decrypt}$ n times for each query. We can see such a naive method takes too much computation overhead for the cloud server. Here, we propose a binary tree based file management method for the access structure computation which greatly reduces the server's computation overhead. As a toy example shown in Fig. 3, let {*A, B, C, D*} be an entire attributes set, then the pointer to a file with $T_{decrypt} = \{(A \wedge B) \vee (C \wedge D)\}$ can be located under the leaf nodes of {*1100*} and {*0011*} of the binary tree. We assume such a file locating process is done by the server when each encrypted file is uploaded to the cloud. Then, if a query constructed by a user with $T_{self-sign} = \{(A \wedge B)\}$ (which is expressed as {*1100*}) is sent to the server, with the binary tree based method (Fig. 3), the cloud server can easily identify the corresponding pointers to all decryptabled files (or says, identify the search cope) under the nodes of {*0000*}, {*0100*}, {*1000*}, {*1100*}. Under such a binary tree based file management, for a user whose number of attributes is k, the cloud server can identify his decryptabled files by collecting file pointers under 2^k leaf nodes. We say the computation overhead of such execution is much smaller compared with a naive method.

Next, we theoretically simulated the performance of the proposed scheme. To show the key feature of our

Table 3 Computation complexity (file processing on the user client side).

Operations	Protocols	Computation Complexity
Create a file (*CreateFile()*)	Ion *et al.* [9]	$O(E_1 \times \log p) + O(\|U\| \times E_0 \times \log p)$
	Li *et al.* [12]	$O(E_1 \times \log p) + O(\|U\| \times E_0 \times \log p)$
	Our Scheme	$O(E_1 \times \log p) + O(l \times E_0 \times \log p)$
Read a file (*Read()*)	Ion *et al.* [9]	$O(\|U\| \times L) + O(\|U\| \times E_0 \times \log p)$
	Li *et al.* [12]	$O(\|U\| \times L) + O(\|U\| \times E_1 \times \log p)$
	Our Scheme	$O(l \times L) + O(\|U\| \times E_1 \times \log p)+$ $O(l \times E_0 \times \log p)$
Update a file (*Update()*)	Ion *et al.* [9]	Not supported
	Li *et al.* [12]	$O(E_1 \times \log p) + O(\|U\| \times E_0 \times \log p) + C_{sign} + C_{enc}$
	Our Scheme	$O(E_1 \times \log p) + O(l \times E_0 \times \log p)$

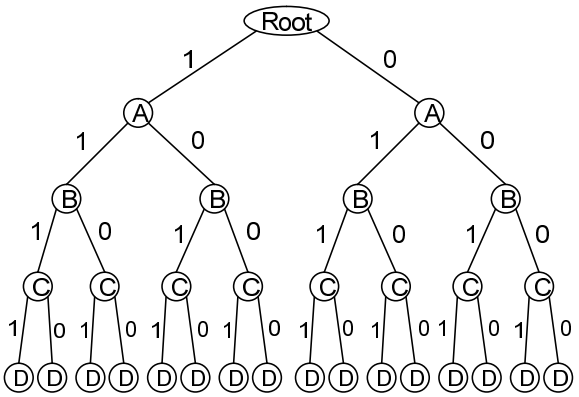

Fig. 3 Binary tree based file management of attribute set.

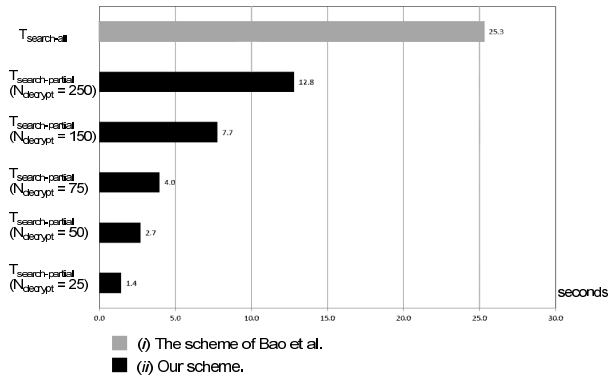

Fig. 4 Simulation of runtime comparison for *Search()* of cases (*i*) and (*ii*) based on the parameter setting: $N_{file} = 500$.

scheme, we adopt a contrastive simulation between an existing multi-user search scheme and ours:

(i) An existing search protocol proposed by Bao *et al.* [2], [23], which is an encrypted keyword search scheme without considering access right.
(ii) Our scheme. Cloud server performs the keyword search while being aware of the user's access right.

Our theoretical performance simulation is based on the following settings. The number of encrypted files on the cloud server: $N_{file} = 500$ (Fig. 4) and $N_{file} = 1000$ (Fig. 5), the average number of keywords for each encrypted file: $N_{keyword} = 10$, the number of attributes in the access structure $T_{self-sign}$ for generating the *ABS*: $N_{attribute} = 10$, let $N_{decrypt}$ be the decryptable files number of the user who generates the query. Recalling the *Search()* phase in Sect. 5.2, we note that to execute the $Search()$ without considering the *ABS* based scope narrowing, the cloud server computes one *paring* and one *HMAC* for each index attached to the encrypted file. We measured the time of *HMAC* using the *OpenSSL toolkit 1.0.1* under a VMware environment with the similar benchmark setting of Sect. 7.2.1, and it costs $t_{hmac} = 0.0041ms$[†]. Then, we implemented the access structure computation, $T_{self-sign} \models T_{decrypt}$, using *C* language. The maximum attribute number of an access structure is set as $N_{attribute} = 10$. To obtain the average computation time t_{binary}, we measured the time of $T_{self-sign} \models T_{decrypt}$

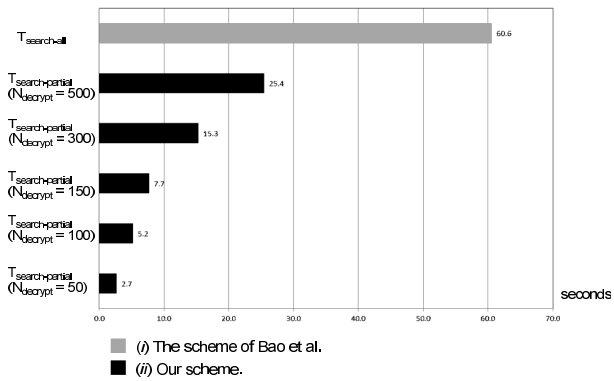

Fig. 5 Simulation of runtime comparison for *Search()* of cases (*i*) and (*ii*) based on the parameter setting: $N_{file} = 1000$.

within the range $1 \le n \le N_{attribute}$[††], the result is $t_{binary} = 0.00027ms$. Following the criterion of Sect. 7.2.1, an *ABS-Verification* approximately costs $t_{verify-abs} = 169ms$.

Finally, we conclude the time for both cases (*i*) and (*ii*) as we described above.

(i) $T_{search-all} = N_{file} \times N_{keyword} \times (t_{paring} + t_{hmac})$
(ii) $T_{search-partial} = t_{verify-abs} + N_{file} \times t_{binary} + N_{decrypt} \times N_{keyword} \times (t_{paring} + t_{hmac})$

We simulate both cases by taking a sample of $N_{decrypt}$ from the following two sets: {*25, 50, 75, 150, 250*} and

[†]HMAC algorithm takes a *16[bytes]* random number as its input, and we choose a secret key of *256[bits]*.

[††]Each attribute in the access structure is *32[bytes]* in our implementation.

{*50, 100, 150, 300, 500*}. It means that the number of decryptable files is assumed to vary separately from *25* to *250* for $N_{file} = 500$, and from *50* to *500* for $N_{file} = 1000$. Figures 4, 5 contrastively show the simulation results for both cases: the execution time $T_{search-partial}$ of case (*ii*) for searching a keyword through the user's decryptable file group is obviously improved than the execution time $T_{search-all}$ of case (*i*) for searching through all files on the cloud server. Our scheme is shown quite efficient because it narrows the search scope and minimize those extra computation (e.g. *paring* and *HMAC* in the $Search()$ phase) from unreadable files.

Remark. From the simulation results we can see that the performance of the *Search()* execution is roughly proportional to the performance of the *paring computation*. For cloud servers equipped with high-performance CPUs and the parallel processing architecture, the runtime for *paring computations* can be improved, and moreover, multiple *paring computations* can be parallelly-processed in less time.

8. Conclusion and Future Work

In this paper, we interdependently harmonized the access control approach to the encrypted keyword search, and proposed the fine-grained access control aware keyword search. The security of the proposed scheme is proved based on several newly extended or defined security requirements. Contributions of our work is clearly shown by a comparison study with several latest existing works. Finally, an efficient implementation method of the proposed scheme is given for reducing computation overhead. As the contrastive performance simulation results, owing to narrowing the server's search scope to the user's decryptable subset, our scheme decreases information leakage from the keyword search progress, and is shown to be efficient.

Among the discussion of Sect. 5.1.2 and Sect. 3 (Definition 3), we argued that the following problem, where a revoked user can still read (decrypt) his previously decryptable ciphertexts with old keys because his attributes are not revoked, is one of our next research directions. We would like to discuss the solution and try to give an extended scheme that protects the proposed cloud system from such a scenario. Exploiting the attribute-based encryption of Sahai *et al.* [18] (which provides attribute revocation) together with our scheme would be the most promising and could be considered in future work.

Acknowledgments

Authors would like to thank Professor *Kazuo Ohta*, Professor *Yoshiaki Hori*, Dr. *Goichiro Hanaoka*, and Professor *Robert H. Deng* for their valuable and suggestive comments.

This work is partially supported by Grant-in-Aid for Young Scientists (B) (23700021), Japan Society for the Promotion of Science (JSPS). This work is also partially supported by Kurata Grant from The Kurata Memorial Hitachi Science and Technology Foundation.

References

[1] J. Benaloh, M. Chase, E. Horvitz, and K. Lauter, "Patient controlled encryption: Ensuring privacy in medical health records," ACM CCSW 2009, pp.103–114, 2009.

[2] F. Bao, R.H. Deng, X. Ding, and Y. Yang, "Private query on encrypted data in multi-user settings," Proceedings of the 4th International Conference on Information Security Practice and Experience Conference, LNCS4991, pp.71–85, 2008.

[3] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," Proceedings of the IEEE Symposium on Security and Privacy, pp.321–334, 2007.

[4] D. Boneh, G. di Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," Proceedings of the Eurocrypt, LNCS 3027, pp.506–522, 2004.

[5] M. Chase and S.S. Chow, "Improving privacy and security in multi-authority attribute-based encryption," Proceedings of the ACM Conference on Computer and Communications Security, pp.121–130, 2009.

[6] R. Curtmola, J.A. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: improved definitions and efficient constructions," Proceedings of the ACM workshop on Cloud Computing Security, pp.79–88, 2006.

[7] D. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," Proceedings of the IEEE Symposium on Security and Privacy, pp.44–55, 2000.

[8] P. Golle, J. Staddon, and B. Waters, "Secure conjunctive keyword search over encrypted data," Proceedings of the Applied Cryptography and Network Security Conference, LNCS 3089, pp.31–45, 2004.

[9] M. Ion, G. Russello, and B. Crispo, "Supporting publication and subscription confidentiality in pub/sub networks," The 6th International ICST Conference on Security and Privacy in Communication Networks, pp.272–289, 2010.

[10] A. Guillevic, Comparing the Pairing Efficiency over Composite-Order and Prime-Order Elliptic Curves, Cryptology ePrint Archive, 2013. http://eprint.iacr.org/2013/218.pdf

[11] S. Kamara and K. Lauter, "Cryptographic cloud storage," Proceedings of Financial Cryptography: Workshop on Real-Life Cryptographic Protocols and Standardization, pp.136–149, 2010.

[12] M. Li, S. Yu, K. Ren, and W. Lou, "Securing personal health records in cloud computing: Patient-centric and fine-grained data access control in multi-owner settings," The 6th International Conference on Security and Privacy in Communication Networks, pp.89–106, 2010.

[13] M. Li, S. Yu, N. Cao, and W. Lou, "Authorized private keyword search over encrypted data in Cloud computing," Proceedings of the 31st Int'l Conference on Distributed Computing Systems, pp.383–392, 2011.

[14] H. Maji, M. Prabhakaran, and M. Rosulek, "Attribute-based signatures," Proceedings of the Cryptographer's Track at RSA Conference 2011 (CT-RSA 2011), LNCS 6558, pp.376–392, 2011.

[15] M. Mihaljevic, M. Fossorier, and H. Imai, "Security evaluation of certain broadcast encryption schemes employing a generalized time-memory-data trade-off," IEEE Commun. Lett., vol.11, no.12, pp.988–990, Dec. 2007.

[16] M. Mihaljevic, "Key management schemes for stateless receivers based on time varying heterogeneous logical key hierarchy," ASIACRYPT 2003, LNCS 2894, pp.137–154, 2003.

[17] NIST Special Publication 800-107. Recommendation for Applications Using Approved Hash Algorithms, 2012.

[18] A. Sahai, H. Seyalioglu, and B. Waters, "Dynamic credentials and ciphertext delegation for attribute-based encryption," Advances in Cryptology - CRYPTO 2012, LNCS 7417, pp.199–217, 2012.

[19] R. Steinwandt and A.S. Corona, "Attribute-based group key establishment," http://eprint.iacr.org/2010/235

[20] K. Tomida, M. Mohri, and Y. Shiraishi, "Keyword searchable encryption with access control from a certain identity-based encryption," Future Information Technology 2014, LNEE 276, pp.113–118, 2014.
[21] C. Wang and C. Hsu, "Integration of hierarchical access control and keyword search encryption in Cloud computing environment," International Journal of Computer and Communication Engineering, vol.2, no.3, pp.333–337, 2013.
[22] W. Wang, Z. Li, R. Owens, and B. Bhargava, "Secure and efficient access to outsourced data," Proceedings of the ACM workshop on Cloud Computing Security, pp.55–65, 2009.
[23] Y. Yang, X. Ding, R.H. Deng, and F. Bao, "Multi-user private queries over encrypted databases," International Journal of Applied Cryptography archive, vol.1, no.4, pp.309–319, Aug. 2009.
[24] M. Zhang, B. Yang, and T. Takagi, "Reconciling and improving to multi-receiver signcryption protocols with threshold decryption," in Security and Communication Networks (SCN), vol.5, no.12, pp.1430–1440, Wiley, 2012.
[25] F. Zhao, T. Nishide, and K. Sakurai, "Realizing fine-grained and flexible access control to outsourced data with attribute-based cryptosystems," The 7th Information Security Practice and Experience Conference (ISPEC), LNCS 6672, pp.83–97, 2011.
[26] F. Zhao, T. Nishide, and K. Sakurai, "Multi-user keyword search scheme for secure data sharing with fine-grained access control," The 14th Annual International Conference on Information Security and Cryptology (ICISC), LNCS 7259, pp.406–418, Springer, 2012.

Appendix A: Detail Algorithms of CP-ABE and ABS

A.1 Bilinear Map

Bilinear map is the basis to understand both CP-ABE and ABS.

Let $\mathbb{G}_0$ and $\mathbb{G}_1$ be two bilinear groups of prime order p. Let $e : \mathbb{G}_0 \times \mathbb{G}_0 \to \mathbb{G}_1$ denote the bilinear map. Let g be a generator of $\mathbb{G}_0$. Bilinear map e has following properties:

- Bilinearity: for all $u, v \in G_0$ and $a, b \in \mathbb{Z}_p$, we have $e(u^a, v^b) = e(u, v)^{ab}$
- Non-degeneracy: $e(g, g) \neq 1$.
- Computable: $e(u, v)$ can be efficiently computed for any $u, v \in G_0$.

A.2 CP-ABE

The construction of our scheme is partially built on the Ciphertext-Policy Attribute-Based Encryption (CP-ABE) [3] of Bethencourt *et al*. We describe the detailed algorithms in this section. At a mathematical level, access structures in CP-ABE are described by a monotone access structure (or access tree) $T_{decrypt}$ [3]. If a set of attributes U satisfies the access tree $T_{decrypt}$, we denote it as $T_{decrypt}(U) = 1$. The function $attr(x)$ denotes the attribute associated with the leaf node x in the tree. The concept of *access tree* and *satisfying an access tree* are used in the whole paper:

Definition 5: *Access Tree*. Let T be a tree representing an access structure. Each non-leaf node of the tree represents a threshold gate, described by its children and a threshold value. If num_x is the number of children of a node x and k_x is its threshold value, then $0 < k_x \leq num_x$. When $k_x = 1$, the threshold gate is an *OR* gate and when $k_x = num_x$, it is an *AND* gate. Each leaf node of the tree simply represents an attribute.

Definition 6: *Satisfying an Access Tree*. Let T be an access tree with root r. Denote by T_x the subtree of T rooted at the node x. Hence T is the same as T_r. If a set of attributes γ satisfies the access tree T_x, we denote it as $T_x(\gamma) = 1$. We compute $T_x(\gamma)$ recursively: if x is a non-leaf node, evaluate $T_{x'}(\gamma)$ for all children x' of node x. $T_x(\gamma)$ returns 1 if and only if at least k_x children return 1. If x is a leaf node, then $T_x(\gamma)$ returns 1 if and only if $att(x) \in \gamma$.

CP-ABE algorithms are described as following steps:

Setup is probabilistic and run by TA. A master key MK and a public key PK are generated in this step.

Let $\mathbb{G}_0$ and $\mathbb{G}_1$ be two bilinear groups of prime order p. Let $e : \mathbb{G}_0 \times \mathbb{G}_0 \to \mathbb{G}_1$ denote the bilinear map. Let g be a generator of $\mathbb{G}_0$. Next it will choose two random exponents $\alpha, \beta \in \mathbb{Z}_p$, and computes:

$$h := g^\beta, f := g^{1/\beta}, Y := e(g, g)^\alpha$$

H is the hash function: $H : \{0, 1\}^* \to \mathbb{G}_0$. So the public key is: $PK := (g, h, f, Y, H)$, and the master key is $MK := (\beta, g^\alpha)$.

Encryption$(PK, m, T_{decrypt})$ is probabilistic and run by a user who wants to encrypt a plaintext message m for a user with a set of attributes in the access structure $T_{decrypt}$, this algorithm generates a ciphertext CT.

It first converts $T_{decrypt}$ to its corresponding monotone span program $M \in (\mathbb{Z}_p^{l \times t})$. Then it randomly chooses $s, u_2, \ldots, u_t \in \mathbb{Z}_p$ and sets $\overrightarrow{u} := (s, u_2, \ldots, u_t)$, $(s_1, \ldots, s_l) := M \cdot \overrightarrow{u}$. The function $\rho(i)$ denotes the attribute associated with ith row of n. Then it computes:

$$c_0 := m \cdot Y^s, c' := h^s, \{c_i := g^{s_i}, c_i' := H(\rho(i))^{s_i}\}_{i=1,\ldots,l}$$

The ciphertext is $CT := (M, c_0, c', \{c_i, c_i'\}_{i=1,\ldots,l})$.

Key-Generation(MK, U) is probabilistic and run by TA: on input the master key MK and a set of attributes U belonging to a user, a secret key SK for these attributes is generated.

With inputs MK and U, it first chooses $r, r_j \in \mathbb{Z}_p$ ($j \in U$), then computes:

$$D := g^{\frac{\alpha+r}{\beta}}, \{D_j := g^r H(j)^{r_j}\}_{j \in U}, \{D_j' := g^{r_j}\}_{j \in U}$$

Then the key is set as $SK := (D, \{D_j, D_j'\}_{j \in U})$. Collusion attack will not work since the blinding value r is used to randomize each user's private key.

Decryption(CT, SK) is deterministic and run by a user with a set of attributes U. On input CT and SK, this algorithm outputs the underlying plaintext m, if CT is a valid encryption of m and U satisfies the access structure $T_{decrypt}$ specified in the computation of CT. Otherwise an error will be returned. There exist λ_i's that satisfy $\sum_{i=1}^{l} \lambda_i \cdot \overrightarrow{M}_i =$

$(1, 0, \ldots, 0)$ where $\overrightarrow{M}_i$ denotes the i-th row vector of M. Then, it computes:

$$\begin{aligned} m' &= c_0 \cdot e(c', D)^{-1} \cdot \prod_{\rho(i)\in U} \left(\frac{e(D_{\rho(i)}, c_i)}{e(D'_{\rho(i)}, c'_i)} \right)^{\lambda_i} \\ &= c_0 \cdot e(c', D)^{-1} \cdot \prod_{\rho(i)\in U} \left(\frac{e(g^r H(\rho(i))^{r_i}, g^{s_i})}{e(g^{r_i}, H(\rho(i))^{s_i})} \right)^{\lambda_i} \\ &= c_0 \cdot e(c', D)^{-1} \cdot \prod_{\rho(i)\in U} e(g, g)^{r \cdot s_i \cdot \lambda_i} \\ &= c_0 \cdot e(h^s, g^{\frac{\alpha+r}{\beta}})^{-1} \cdot e(g, g)^{r \cdot \sum_{\rho(i)\in S} s_i \cdot \lambda_i} \\ &= m \cdot e(g, g)^{\alpha s} \cdot e(g, g)^{-\alpha s - rs} \cdot e(g, g)^{rs} \\ &= m \end{aligned}$$

A.3 ABS

Our scheme is also based on the Attribute-Based Signature (ABS) [14] of Maji *et al*. We describe the detailed algorithms of ABS in this section. There are two entities exist in *ABS*: a central trusted authority (TA) and users. The authority is in charge of the users' cryptographic keys. Denote the universe of attributes as U, As the access structure in the *CP-ABE*, there is a monotone boolean claim-predicate (access structure) T_{sign} over U whose inputs are associated with attributes of U. We say that an attribute set U satisfies a predicate T_{sign} if $T_{sign}(U) = 1$. The algorithms are described as follows.

Setup The authority obtains a key pair (PK, MK) and outputs public parameters PK and keeps a private master key MK.

Choose suitable cyclic groups $\mathbb{G}$ and $\mathbb{H}$ of prime order p, equipped with a bilinear pairing $e : \mathbb{G} \times \mathbb{H} \to \mathbb{G}_T$. Choose a hash function $\mathcal{H} : \{0, 1\}^* \to \mathbb{Z}_p^*$. We treat $\mathbb{A} = \mathbb{Z}_p^*$ as the universe of attributes, where p is the size of the cyclic group. t_{max} means the claim-predicate whose monotone span program has width at most t_{max}. Choose random generators:

$$g, C \leftarrow \mathbb{G}; \quad h_0, \ldots, h_{t_{max}} \leftarrow \mathbb{H}$$

Choose random $a_0, a, b \leftarrow \mathbb{Z}_p^*$ and set:

$$A_0 = h_0^{a_0}; \quad A_j = h_j^a \text{ and } B_j = h_j^b (\forall j \in t_{max})$$

The master key is $MK = (a_0, a, b)$. The public key PK is a description of the groups $\mathbb{G}$, $\mathbb{H}$ and their pairing function, as well as:

$$(\mathcal{H}, g, h_0, \ldots, h_{t_{max}}, A_0, \ldots, A_{t_{max}}, B_0, \ldots, B_{t_{max}}, C)$$

Key-Generation(MK, U) To assign a set of attributes U to a user, the authority computes a signing key SK_U and gives it to the user.

On input MK as above and attribute set $U \subseteq \mathbb{A}$, Choose random generator $K_{base} \in \mathbb{G}$. Then Set: $K_0 = K_{base}^{1/a_0}$; $K_u = K_{base}^{1/(a+bu)}$, $(\forall u \in U)$.

$$SK_U = (K_{base}, K_0, \{K_u | u \in U\})$$

Sign(PK, SK_U, m, T_{sign}) To sign a message m with a claim-predicate T_{sign}, and a set of attributes U such that $T_{sign}(U) = 1$, the user computes a signature σ by (PK, SK_U, m, T_{sign}).

First, convert T_{sign} to its corresponding monotone span program $M \in (\mathbb{Z}_p)^{l \times t}$, with row labeling $u : [l] \to \mathbb{A}$. Also compute the vector $\vec{v}$ that corresponds to the satisfying assignment U. Compute $\mu = \mathcal{H}(m \| T_{sign})$, then pick random $r_0, r_1, \ldots, r_l$ and compute:

$$Y = K_{base}^{r_0}; \quad S_i = (K_{u(i)}^{v_i})^{r_0} \cdot (Cg^{\mu})^{r_i}, (\forall i \in l);$$

$$W = K_0^{r_0}; \quad P_j = \prod_{i=1}^{l} (A_j B_j^{u(i)})^{M_{ij} \cdot r_i}, (\forall j \in t).$$

Here, the signer may not have $K_{u(i)}$ for every attribute $u(i)$ mentioned in the claim-predicate. But when this is the case $v_i = 0$, and so the value is not needed. The signature is $\sigma = (Y, W, S_1, \ldots, S_l, P_1, \ldots, P_l)$

Verify$(PK, m, T_{sign}, \sigma)$ To verify a signature σ on a message m with a claim-predicate T_{sign}, a user runs $Verify(PK, m, T_{sign}, \sigma)$, which outputs a boolean value, accept or reject.

First, convert T_{sign} to its corresponding monotone span program $M \in (\mathbb{Z}_p)^{l \times t}$, and compute $\mu = \mathcal{H}(m \| T_{sign})$, if $Y = 1$, then $reject$. Otherwise check the following constraints:

$$e(W, A_0) \stackrel{?}{=} e(Y, h_0);$$

$$\prod_{i=1}^{l} e(S_i, (A_j B_j^{u(i)})^{M_{ij}}) \stackrel{?}{=} \begin{cases} e(Y, h_1) e(Cg^{\mu}, P_1), (j = 1); \\ e(Cg^{\mu}, P_j), (j > 1); \end{cases}$$

for $1 \le j \le t$. Returns $accept$ if all the above checks succeed, and $reject$ otherwise.

Appendix B: Security Proofs

Proof of Theorem 1. It suffices for us to construct a *PPT* simulator $\mathcal{A}^*$ such that for all $t \in \mathbb{N}$, for all *PPT* adversaries $\mathcal{A}$, all functions f, given the T_t, $\mathcal{A}^*$ can simulate $\mathcal{A}(V_t)$ with non-negligible probability. More specifically, we show that $\mathcal{A}^*$ with T_t can generate a view V_t^* which is computationally indistinguishable from V_t, the actual view of $\mathcal{A}$. Next we discuss both $t = 0$ and $t > 0$.

If $t = 0$, then $Q_t = \emptyset$, $A_t = \emptyset$, $Sig(Q_t) = \emptyset$. $\mathcal{A}^*$ builds $V_t^* = \{CT^*, I(w)^*\}$ from random elements. It is easy to check that V_t^* and V_t are computationally indistinguishable if *HMAC* is unforgeable (for generating $I(w)$ in the Cap_w) and the CT based on the *CP-ABE* [3] is secure. Recall that the generation of $I(w)$ contains a random number generation and a *HMAC* computation on that random number each time, so all $I(w)$ generated from the same keyword are different from each other and the *HMAC* is also infeasible to be forged.

If $t > 0$, $\mathcal{A}^*$ builds $V_t^* = \{CT^*, I(w)^*, Q_t^*, Sig(Q_t)^*, A_t^*\}$. To be general, we suppose that all queries q in Q_t are from

distinct users, but some of them may query the same keywords. (*i*) Discussion of CT^* and $I(w)^*$ are almost the same as the case of $t = 0$. (*ii*) For Q_t^* and Q_t, recall that the generation of Q_t is decided by both $h(w)$ and $K_{U_{id}}$: $Q(w) = h(w)^{K_{U_{id}}}$. $\mathcal{A}^*$ first generates a simulated complementary key set $\{x_1^*, \ldots, x_u^*\}$ for all entries which $x_u^* \in \mathbb{Z}_p^*$. $\mathcal{A}^*$ selects a simulated complementary key element from the set $\{x_1^*, \ldots, x_u^*\}$, say x_i^* for u_i^*. Then, $\mathcal{A}^*$ selects a random element element $r_g \in \mathbb{G}_0$, and it can generate a simulated $Q_i^* = r_g^{x_i^*}$. We can see an actual query $Q(w) = h(w)^{K_{U_{id}}}$ and a simulated query $Q_u^* = r_g^{x_u^*}$ are computationally indistinguishable if $h()$ is a pseudorandom function. (*iii*) For $Sig(Q_t)^*$ and $Sig(Q_t)$, recall that $Sig(Q_t)$ is generated from the Q_t as $Sig(Q_t) = Sign\{PK_S, SK_S, h(Q_t), T_{self-sign}\}$. $\mathcal{A}^*$ selects a simulated $SK^* \in \mathbb{G}$, together with the simulated $Q^* = r_g^{x^*}$ as we described in (*ii*), and he generates $h(Q^*)$ and $Sig(Q_t)^*$. It is easy to see an an actual $Sig(Q_t)$ and a simulated $Sig(Q_t)^*$ are computationally indistinguishable if $h()$ is a pseudorandom function. (*iv*) Finally, for A_t^* and A_t, given the above indistinguishability results, the indistinguishability between A_t^* and A_t is straightforward. □

Proof of Theorem 2. To prove this theorem, it suffices for us to state that if there exists a PPT adversary $\mathcal{A}$ that breaks the query unforgeability of our protocol defined in Definition 2 with an advantage ϵ, then there exists a PPT adversary $\mathcal{B}$ that can first forge a query $Q(w)$ for a target keyword w, and moreover, $\mathcal{B}$ can succeed in forging the digital signature, *ABS*, for the forged $Q(w)$ with the same amount of advantage. We briefly provide the proof of this theorem which is based on the security proof of *ABS*'s unforgeability in Maji *et al.* [14] (Detailed proof is omitted here).

The detailed proof contains two parts: (*i*) The first part is straightforward: A valid query is generated as $Q(w) = h(w)^{K_{U_{id}}}$. An adversary is infeasible to retrieve $K_{U_{id}}$ without K_{msk} according to $CK_{U_{id}} = g^{K_{msk}/K_{U_{id}}}$, because the master secret key K_{msk} is assumed to be securely managed by TA. (*ii*) Based on the result of *ABS*'s unforgeability of Maji *et al.* [14], an adversary $\mathcal{B}$, without accessing the secret signing key of the target user, $\mathcal{B}$'s probability of successfully generating $Sig(Q)$ from Q, which satisfies $Verify(PK_S, Q, T_{self-sign}, Sig(Q)) = True$, is negligible. Concluding both parts, $\mathcal{B}$'s total advantage for generating the legitimate query set $\{Q, Sig(Q)\}$ must be negligible. This proves the theorem. □

Proof of Theorem 3. The proof is quite straightforward, and we only state the intuition behind the proof. The indexes of the two keywords w_1 and w_2 are $\mathrm{I}(w_1) = [R_1, HMAC_{k_{w_1}}(R_1)]$, and $\mathrm{I}(w_2) = [R_2, HMAC_{k_{w_2}}(R_2)]$, where R_1 and R_2 are random, k_{w_1} and k_{w_2} denote the secret keys generated from w_1 and w_2, respectively. Since the complementary key $CK_{U_{id}}$ of a revoked user is deleted from the user list L, the revoked user can never get k_{w_1} and k_{w_2} from the keywords and the query key $K_{U_{id}}$ it has. Finally, the only way that the revoked user can guess the correct bit of b is trying to reverse the $HMAC_{k_{w_1}}(R_1)$ or $HMAC_{k_{w_2}}(R_2)$ to get information about w_1 and w_2. Based on the preimage resistance security property of *HMAC* [17], we can conclude that $\mathrm{I}(w_1)$ and $\mathrm{I}(w_2)$ are independent of w_1 and w_2, respectively, from the perspective of the revoked user. So the advantage of the adversary guessing the correct bit cannot be significantly different from 1/2. □

Fangming Zhao received the M.E. degree in information engineering from Kyushu University in 2008. He is currently working as a researcher at Computer Architecture & Security System Laboratory, Corporate Research & Development Center, Toshiba Corporation. His research interests include information security and applied cryptography. He is a member of IEICE, CIGRE.

Takashi Nishide received a B.S. degree from the University of Tokyo in 1997, an M.S. degree from the University of Southern California in 2003, and a Dr.E. degree from the University of Electro-Communications in 2008. From 1997 to 2009, he had worked at Hitachi Software Engineering Co., Ltd. developing security products. From 2009 to 2013, he had been an assistant professor in Kyushu University and from 2013 he is an associate professor in University of Tsukuba. His research is in the areas of cryptography and information security.

Kouichi Sakurai received the B.S. degree in mathematics from the Faculty of Science, Kyushu University and the M.S. degree in applied science from the Faculty of Engineering, Kyushu University in 1986 and 1988 respectively. He received D.E. degree from the Faculty of Engineering, Kyushu University in 1993. His current research interests are in cryptography and information security. Dr. Sakurai is a member of the Information Processing Society of Japan, the Mathematical Society of Japan, ACM and the International Association for Cryptologic Research.